Rust at Sentry

7 Years Later

Armin @*mitsuhiko* Ronacher

What's happening?

Who am I

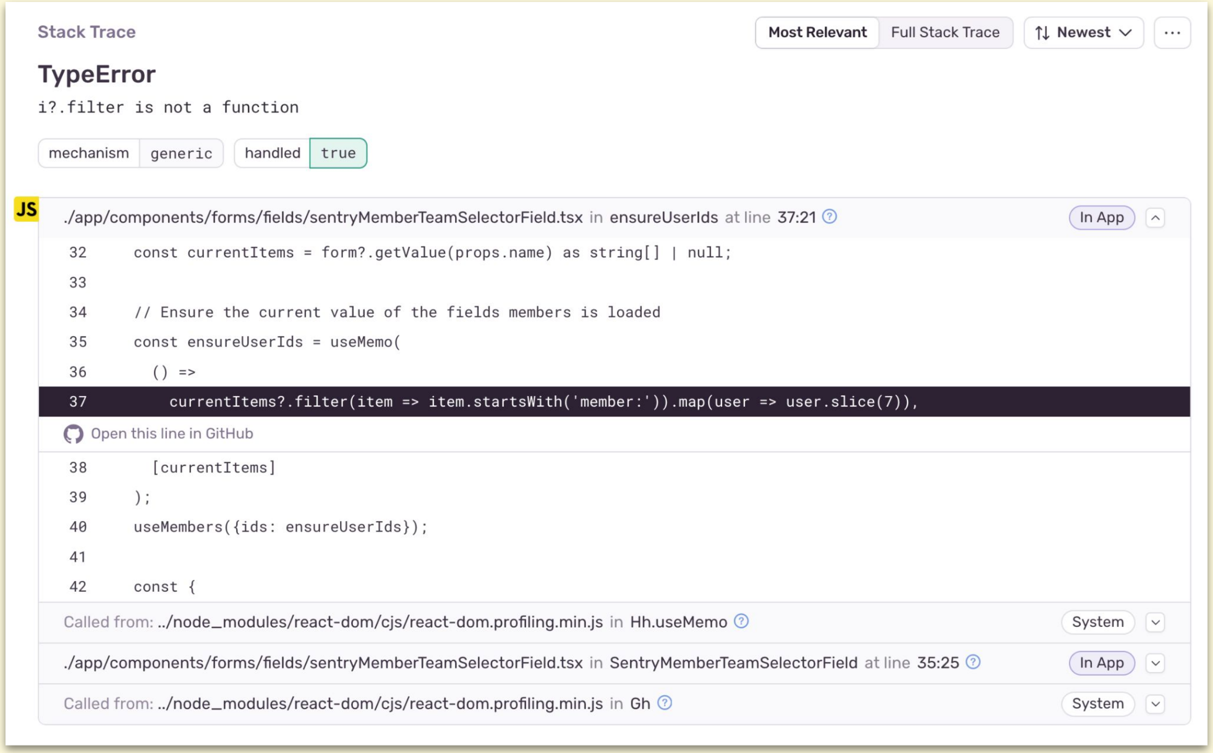
- Armin Ronacher
- @mitsuhiko
- https://lucumr.pocoo.org/
- I love Open Source
- Flask, Insta, Jinja2, MiniJinja, …

What's Sentry

- https://sentry.io/
- Error and Crash Monitoring
- Application Performance Monitoring
- Session Replays etc.
- Open Source (*)
- A Python Shop

**: some is BUSL licensed with a 3 year Apache 2 cliff*

Errors and Crashes



Stack Trace

Most Relevant | Full Stack Trace | Newest

TypeError

```
i?.filter is not a function
```

mechanism `generic` handled `true`

./app/components/forms/fields/sentryMemberTeamSelectorField.tsx in ensureUserIds at line 37:21 — In App

```
  const currentItems = form?.getValue(props.name) as string[] | null;

  // Ensure the current value of the fields members is loaded
  const ensureUserIds = useMemo(
    () =>
      currentItems?.filter(item => item.startsWith('member:')).map(user => user.slice(7)),
    [currentItems]
  );
  useMembers({ids: ensureUserIds});

  const {
```

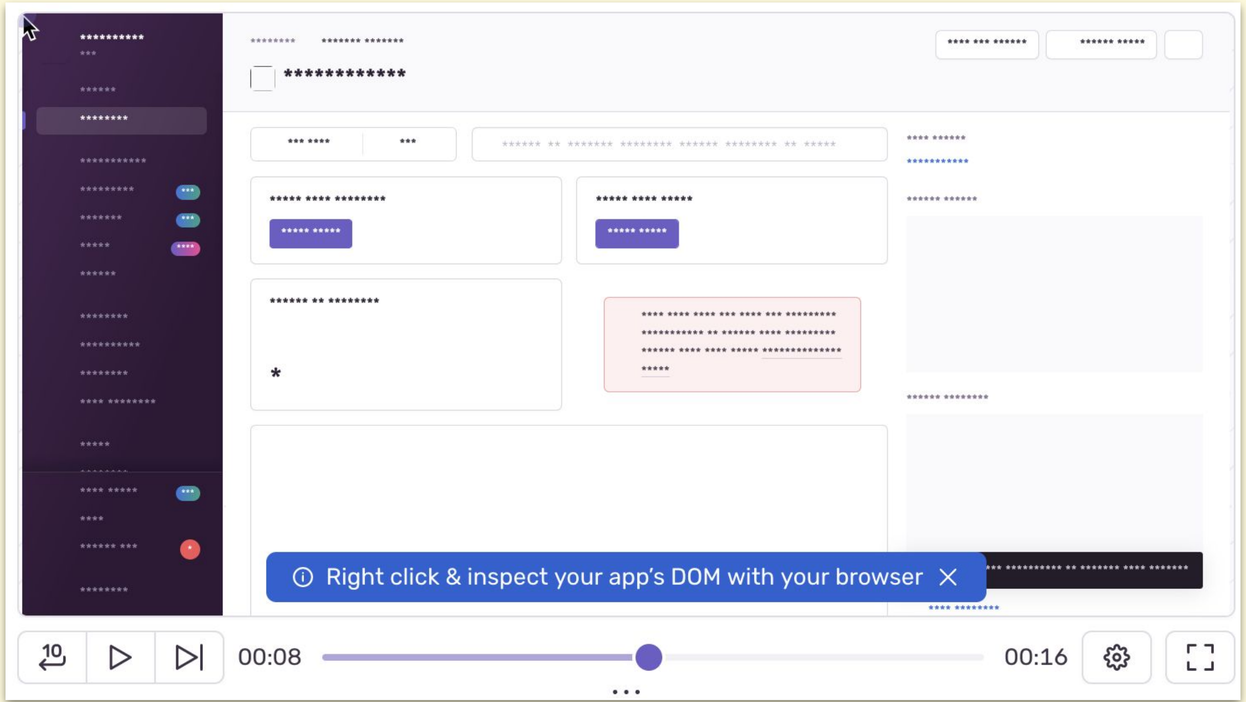
Open this line in GitHub

Called from: ../node_modules/react-dom/cjs/react-dom.profiling.min.js in Hh.useMemo — System

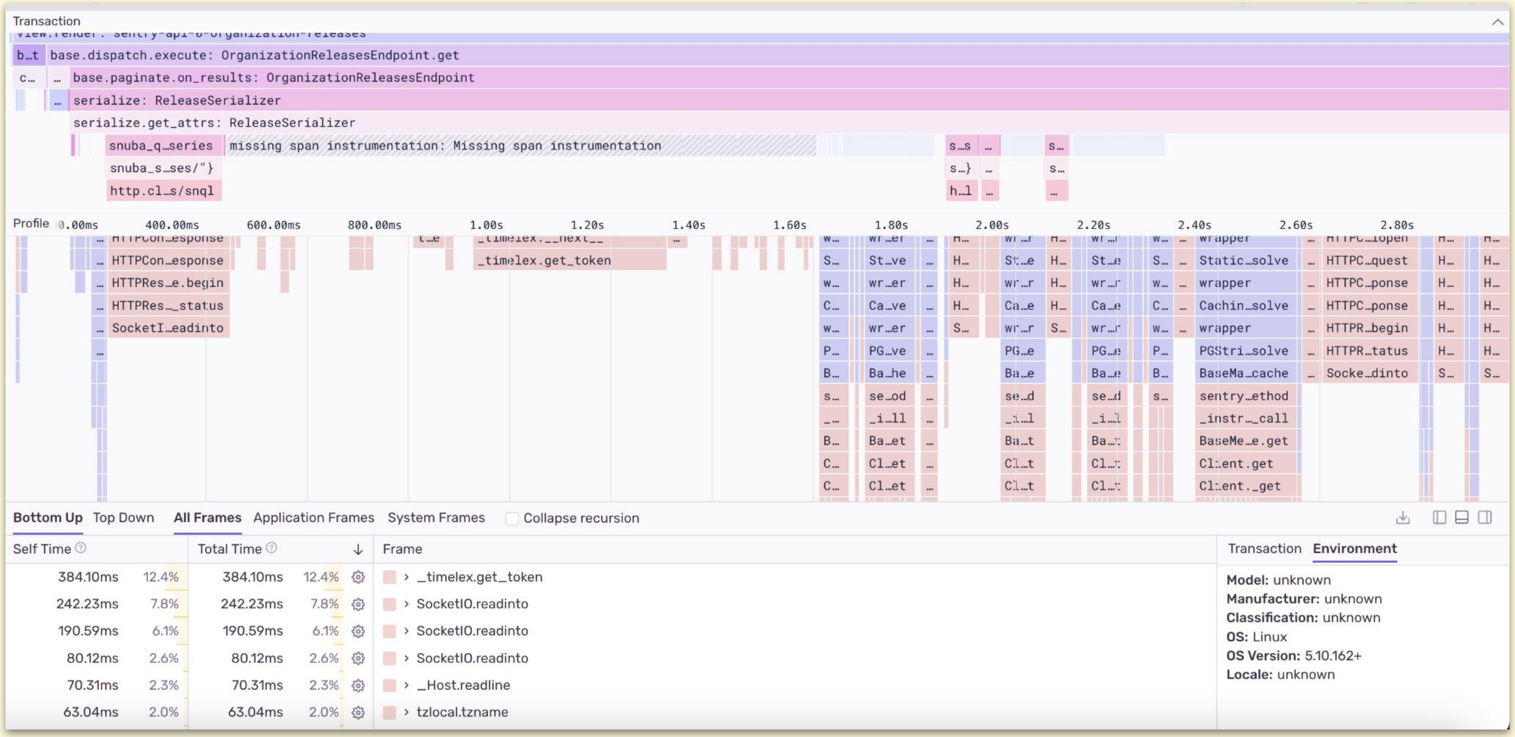
./app/components/forms/fields/sentryMemberTeamSelectorField.tsx in SentryMemberTeamSelectorField at line 35:25 — In App

Called from: ../node_modules/react-dom/cjs/react-dom.profiling.min.js in Gh — System

Replays


Right click & inspect your app's DOM with your browser
00:08
00:16
10

Profiles

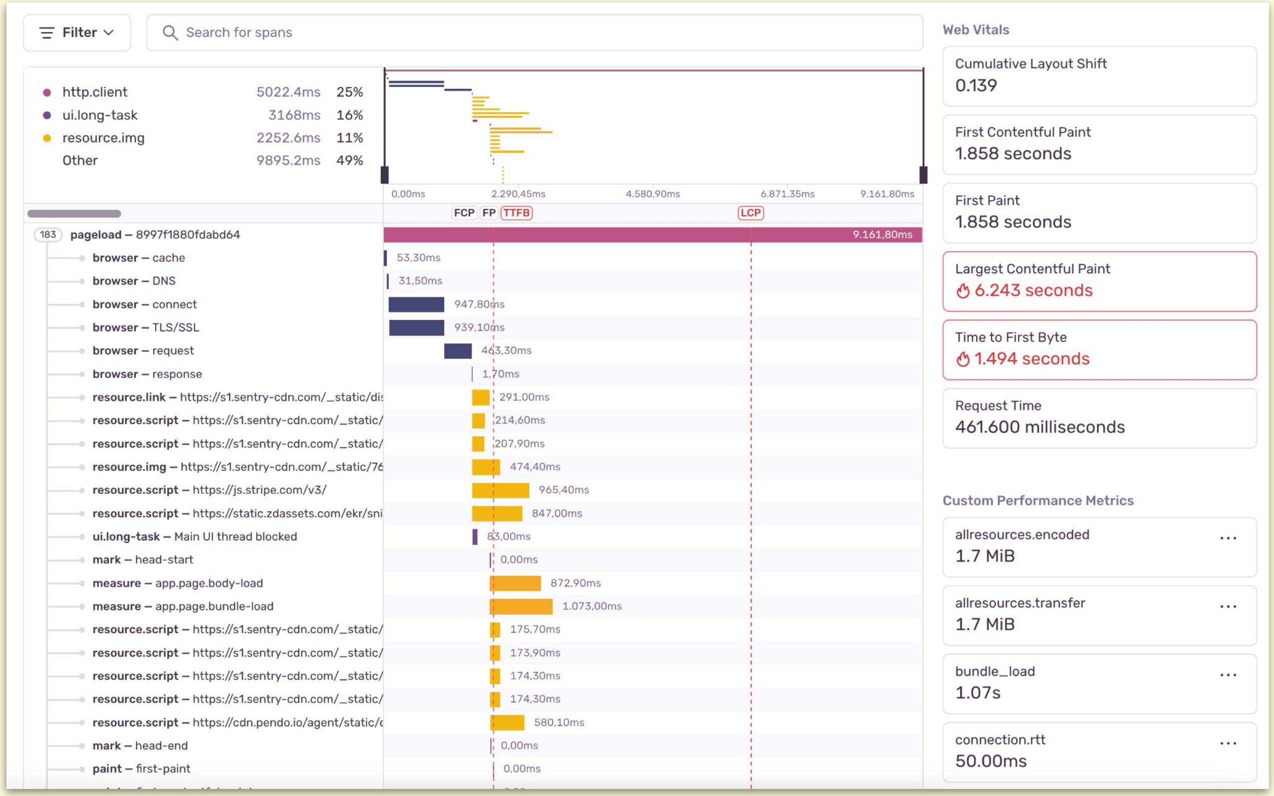


Self Time		Total Time		Frame
384.10ms	12.4%	384.10ms	12.4%	_timelex.get_token
242.23ms	7.8%	242.23ms	7.8%	SocketIO.readinto
190.59ms	6.1%	190.59ms	6.1%	SocketIO.readinto
80.12ms	2.6%	80.12ms	2.6%	SocketIO.readinto
70.31ms	2.3%	70.31ms	2.3%	_Host.readline
63.04ms	2.0%	63.04ms	2.0%	tzlocal.tzname

Transaction Environment

Model: unknown
Manufacturer: unknown
Classification: unknown
OS: Linux
OS Version: 5.10.162+
Locale: unknown

Traces



Filter

Search for spans

● http.client	5022.4ms	25%
● ui.long-task	3168ms	16%
● resource.img	2252.6ms	11%
Other	9895.2ms	49%

183 **pageload** — 8997f1880fdabd64	9.161,80ms
browser — cache	53,30ms
browser — DNS	31,50ms
browser — connect	947,80ms
browser — TLS/SSL	939,10ms
browser — request	463,30ms
browser — response	1,70ms
resource.link — https://s1.sentry-cdn.com/_static/dis	291,00ms
resource.script — https://s1.sentry-cdn.com/_static/	214,60ms
resource.script — https://s1.sentry-cdn.com/_static/	207,90ms
resource.img — https://s1.sentry-cdn.com/_static/76	474,40ms
resource.script — https://js.stripe.com/v3/	965,40ms
resource.script — https://static.zdassets.com/ekr/sni	847,00ms
ui.long-task — Main UI thread blocked	83,00ms
mark — head-start	0,00ms
measure — app.page.body-load	872,90ms
measure — app.page.bundle-load	1.073,00ms
resource.script — https://s1.sentry-cdn.com/_static/	175,70ms
resource.script — https://s1.sentry-cdn.com/_static/	173,90ms
resource.script — https://s1.sentry-cdn.com/_static/	174,30ms
resource.script — https://s1.sentry-cdn.com/_static/	174,30ms
resource.script — https://cdn.pendo.io/agent/static/c	580,10ms
mark — head-end	0,00ms
paint — first-paint	0,00ms

Web Vitals

Cumulative Layout Shift	0.139
First Contentful Paint	1.858 seconds
First Paint	1.858 seconds
Largest Contentful Paint	6.243 seconds
Time to First Byte	1.494 seconds
Request Time	461.600 milliseconds

Custom Performance Metrics

allresources.encoded	1.7 MiB
allresources.transfer	1.7 MiB
bundle_load	1.07s
connection.rtt	50.00ms

Why Rust?

- Initially personal interest
- Was really good for redistribution (sentry-cli)
- Was really nice to expose to Python
- Over time: we quite like it
- Predictable at runtime
- Tooling is really good

A Company's Origin Story is a Legend

- Memory gets foggy over time
- Technology choices are less well informed and more incidental
- Is Jane Street really successful because of OCaml?

Rust @ Sentry Stats

- rust libraries + services: 180kLOC
- Sentry Python Monolith: 455kLOC
- Sentry TypeScript SPA: 612kLOC

Third most popular language by LOC

Why we picked it

Predictable Runtime Behavior

- Feels like Python
- No whacky memory behavior
  - (aside from suffering of fragmentation — hi jemallocator)
- CPU usage mostly stays predictable
- Performs well for a long time

Fits into Python

- Great at extension modules
- For us: cffi + milksnake (do not use!)
- Nowadays: PyO3 + maturin

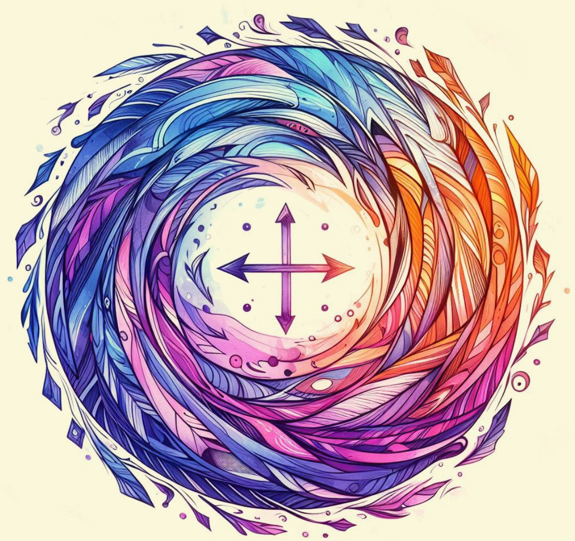
Unexpected Wins

Rust is Outbound

- We quite actively contribute to external crates in Rust
- We rarely do so in Python
- Fork and depend on fork works well!
- **Cargo as tooling changes behavior**

Standardized Tooling

- One code style
- Almost universally embraced lints
- Rather well established patterns
- Jumping between code-bases feels natural
- Moving code between crates is trivial
- Painless compiler upgrades



The DX is Dope

- cargo
- rustup
- rust-analyzer
- docs (std + crate)

Types and Borrow Checker

- Modern Rust makes you a better programmer
- Types for the most part are helpful
- Borrow checker is not too annoying any more
- Makes you suspicious of a lot of Python code

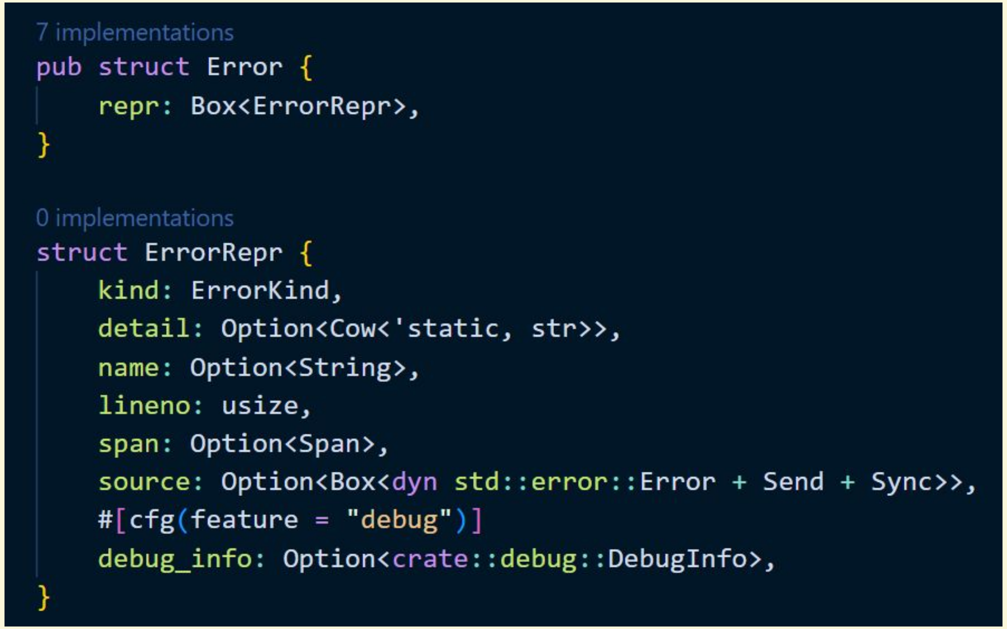
Unexpected Issues

Why is there so much memmove?

- Large error types
- String::clone and friends

Large Result Types (Large Errors)

- The compiler sometimes is bad at optimizing result mapping



```
7 implementations
pub struct Error {
    repr: Box<ErrorRepr>,
}

0 implementations
struct ErrorRepr {
    kind: ErrorKind,
    detail: Option<Cow<'static, str>>,
    name: Option<String>,
    lineno: usize,
    span: Option<Span>,
    source: Option<Box<dyn std::error::Error + Send + Sync>>,
    #[cfg(feature = "debug")]
    debug_info: Option<crate::debug::DebugInfo>,
}
```

Shlemiel the Painter

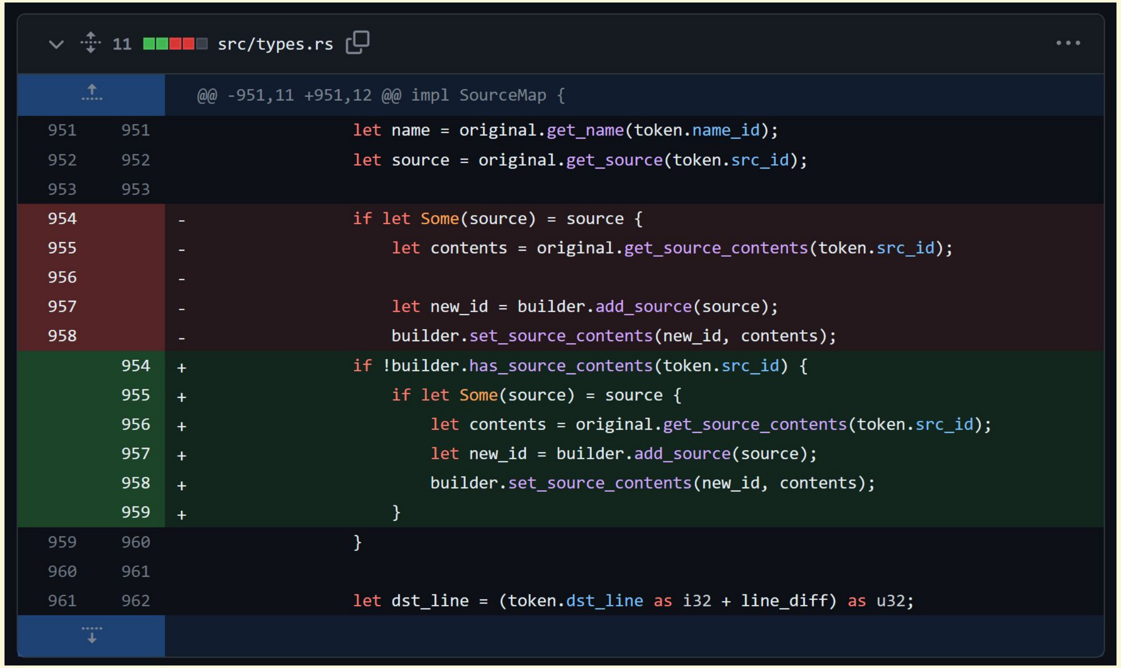
- Work gets progressively harder
- Classic case: cstrings (strcat)
- But also OFFSET + LIMIT in SQL

Rust has a family of performance issues that are related

- Fear of lifetimes cause bad lookups
- String assigns become string clones

Shlemiel Paints the Entire Street For Every Dot

- Add an offset to N tokens, clone entire source for every token



```diff
11 src/types.rs
@@ -951,11 +951,12 @@ impl SourceMap {
                 let name = original.get_name(token.name_id);
                 let source = original.get_source(token.src_id);

-                if let Some(source) = source {
-                    let contents = original.get_source_contents(token.src_id);
-
-                    let new_id = builder.add_source(source);
-                    builder.set_source_contents(new_id, contents);
+                if !builder.has_source_contents(token.src_id) {
+                    if let Some(source) = source {
+                        let contents = original.get_source_contents(token.src_id);
+                        let new_id = builder.add_source(source);
+                        builder.set_source_contents(new_id, contents);
+                    }
                 }

                 let dst_line = (token.dst_line as i32 + line_diff) as u32;
```

Strings are … not optimal

- Maybe we should use more Arc<str>?
- But Arc<str> is not particularly efficient
- String’s extra capacity is odd in public APIs
- Similar issue with Vec<u8> (broadcast to N sockets)

WCB

Errors

- Still no stack trace on std::error::Error
- Errors don’t have names (parsing Debug output)

Life Before Main / Registry

- We would love a supported #[ctor]
- Or a way to register startup functions

Async and Tokio

From Actix to Running our own Show

- Started out with actix + actix-web
- Actor frameworks feel great
- Backpressure management is a giant pain and messy
- Moved from pre-tokio 1.0 to async/await

How I learned to love the async Bomb

- Use less async
- Use More Channels
- Embrace Backpressure
- (Cancellations are still hard)

Rust is Good For Us

Rust Community: Let's talk

Some Thoughts

- Nobody is perfect
- Building things is hard
- Good intentions can still result in bad outcomes
- Rust made it this far, let’s work on it together
- We all are more nuanced in Person